Information Technology Division

ELECTRONICS RESEARCH LABORATORY

GENERAL DOCUMENT
ERL-0631-GD

SHIVA MARK II HARDWARE ARCHITECTURE, VERSION 1

by

D.A. Krnak, A.J.S. Yakovleff, J.D. Yesberg, M.S. Anderson and P.C. Drewer

APPROVED FOR PUBLIC RELEASE

2

AR-006-970

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

Information Technology Division

GENERAL DOCUMENT
ERL-0631-GD

SHIVA MARK II HARDWARE ARCHITECTURE, VERSION 1

by

D.A. Krnak, A.J.S. Yakovleff, J.D. Yesberg, M.S. Anderson and P.C. Drewer

SUMMARY

This document describes the hardware aspects of the Shiva multiprocessor, which has a dynamically reconfigurable architecture and supports heterogeneity. The system is meant to be used as an accelerator for computationally intensive tasks, and is used in conjunction with a workstation.

AUG 92

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory, PO Box 1500, Salisbury, South Australia, 5108.

CONTENTS

FIGURES

DEDICATION

This report is dedicated to David Krnak whose tragic and untimely death saddened his colleagues. David was an intelligent, amiable person likely to make a significant contribution to the research program of DSTO's Information Technology Division. He will be difficult to replace.

Mark Anderson.

1. INTRODUCTION

This document describes the hardware comprising the Shiva Mark II multiprocessor. We do not justify the architectural choices made, nor do we compare the Shiva with other similar architectures or present applications of the Shiva. Such issues are discussed in [2, 3, 4].

The level of detail provided herein is appropriate to that required by a computer architect or a system software engineer. More detailed design information is contained in the schematic diagrams and programmable device source files, while justification for some of the design features can be found in various manuals such as [6,7,8].

The aims of the Shiva project was to design, fabricate and test a high performance multiprocessor computer based on the concepts of heterogeneity (the processing elements need not be identical) and dynamic reconfigurability (the logical data paths can be reconfigured at runtime). The hardware is also to be used as a testbed for research into a variety of multiprocessing software issues such as automatic parallelism extraction from sequential programs.

It was decided that the entire Shiva hardware was too complex to design in one stage and so initially a single processor board was fabricated and tested. This computer, dubbed the Shiva Mark I, allowed the designers to gain experience in designing a large and complex board and also provided a base on which system software could be developed. The hardware design of the Shiva Mark I is detailed in [5].

This second phase of the project involved the design and construction of the Shiva Mark II, a multiprocessor with up to 9 processing elements.

Details of the Shiva system software and of the interface available to the application programmer will be published in a separate document.

Chapter 2 gives an overview of the architecture of the Shiva Mark II and each of its main components. Chapter 3 describes the hardware design of each unit in more detail.

2. ARCHITECTURAL OVERVIEW

2.1 Major Components

The architecture of the Shiva Mark II can be described as a hybrid of bus and pipeline architectures. It consists of a master controller board, hereafter referred to as the *master*, and any number (in principle) of auxiliary, or *slave*, boards. It is significant to note that the slave boards need not be of the same type, implying the possibility of a heterogeneous architecture.

Although the Shiva concept allows for many different types of slave boards, this report will describe only one type of slave that employs an Intel i860 as the main processing element.

Referring to Figure 1, it can be seen that each board, be it master or i860 slave, comprises a memory unit which can be accessed either directly by the resident processor via a *hotline* or through a bus to which each processor has access. The memory units together form a shared address space. Lastly, each slave unit has a direct connection to its neighbour through FIFO registers, thus forming a data pipeline.

Access to the outside world is handled by the master via a RS232 interface (see the section on the subsystem) and an SBus interface which permit transfers to or from a SPARCstation. The slave units do not have direct access to these interfaces. In the same manner, pipeline and hotline connections are private.

Since the different elements are memory mapped, the logical data paths are determined by the software.

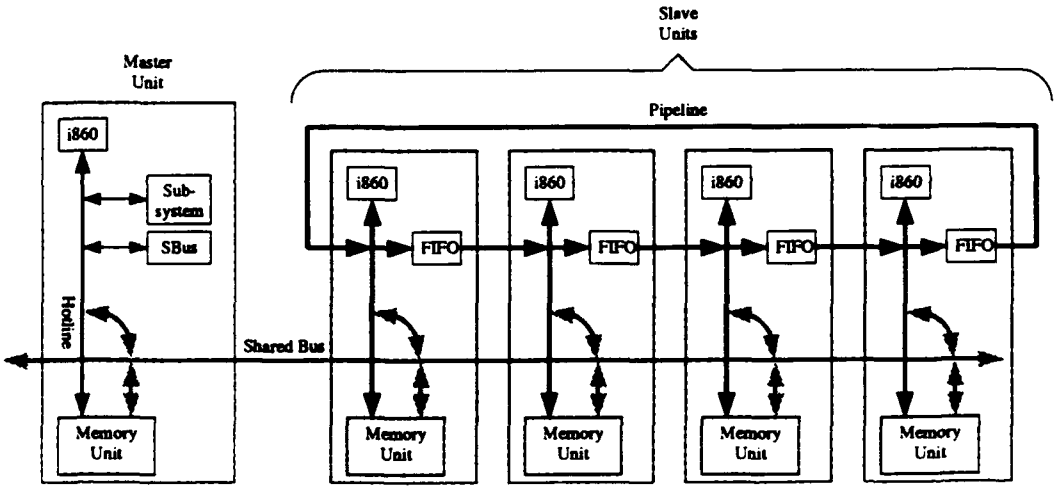


Figure 1. Master and Slave Units data paths

2.2 Master

The master unit contains the following elements:

- coordinator
- memory unit
- SBus interface
- subsystem
  a. bootstrap EPROM
  b. real-time clock
  c. serial interface
  d. read-only/write-only registers
- bus arbitrator

The control signals to and from the i860 are handled by a *coordinator* which includes address/parameter FIFOs to make use of the processor's pipelining capabilities. The coordinator maps requests from the i860 to the various devices (local memory, SBus or subsystem) or to the bus arbitrator if any of the other memory units is to be accessed.

2.3 i860 Slave

The slave unit is essentially a stripped-down version of the master and contains:

1. coordinator
2. memory unit (the same as on the master board)
3. data pipeline

A slightly different version of coordinator maps requests from the i860 to either the pipeline, the local memory unit or the arbitrator (via the bus).

2.4 Other Slave Boards

The Shiva concept is designed to allow many different types of slaves and one alternative slave board that has been proposed is the Neural Accelerator Board [1]. Anderson et al. [2] discusses some applications of such a heterogeneous architecture.

3. COMPONENT DESIGN

3.1 Master Unit

The master board consists of a number of systems or modules which communicate with each other via well-defined handshaking protocols. The modules on the master board will now be discussed.

3.1.1 Coordinator

The coordinator module is the interface between the i860 chip and all the other modules on the same board (including the bus interface). The coordinator is responsible for "catching" requests issued by the i860, determining what other module needs to be accessed, generating the appropriate control signals to activate this module, waiting for the required module to perform its operation and signal completion, and finally signalling to the i860 that the request is complete.

The coordinator determines which modules needs to be accessed by looking at address bits A31..A27. The memory map for the master unit is as shown in Figure 2.

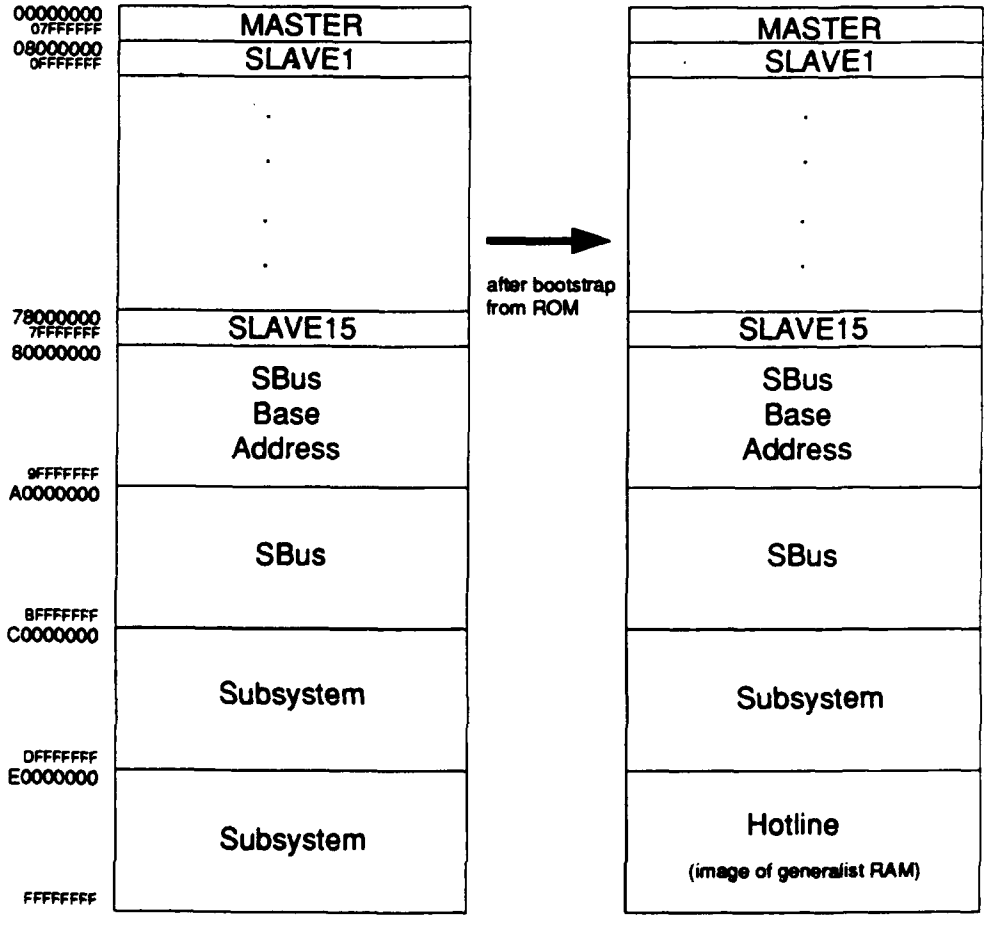


Figure 2. Master Unit memory map

When the master board is powered up, the subsystem appears in its normal location at 0xC0000000 and also at 0xE0000000. This is so that the bootstrap ROM (which is at the end of the subsystem address space) will appear at the end of the logical address space. The first

activity of the i860 after power up is to read an instruction located at address 0xFFFFFF00, which will now be mapped into the bootstrap ROM.

During subsequent operation of the i860, assertion of the INT# pin will cause the i860 to enter an interrupt service routine (ISR) also at address 0xFFFFFF00. Hence, before the first interrupt is received, code for the ISR should be placed at location 0x07FFFF00 (the end of the hotline memory) and then the hotline memory mapped to the end of the logical address space by writing to a special address in the subsystem (see section 3.1.4).

The coordinator incorporates an address FIFO which allows the i860's pipelined bus cycle facility to be utilised. Up to 3 outstanding bus requests can be stored in the address FIFO allowing the i860 to continue processing and maintain optimum speed despite peripheral devices with long latency times.

A block diagram of the master coordinator is shown in Figure 3. Note that there is a request signal going to each of the modules and a corresponding acknowledgement signal, namely, G_MREQ# and G_NAOK_MEM (hotline memory), G_BREQ# and G_NAOK_BUS (global bus), G_SSREQ# and G_NAOK_SUB (subsystem) and G_SBREQ# and G_NAOK_SBUS (SBus interface). G_FLIP is the signal from the subsystem that controls when the address maps are swapped as described above.

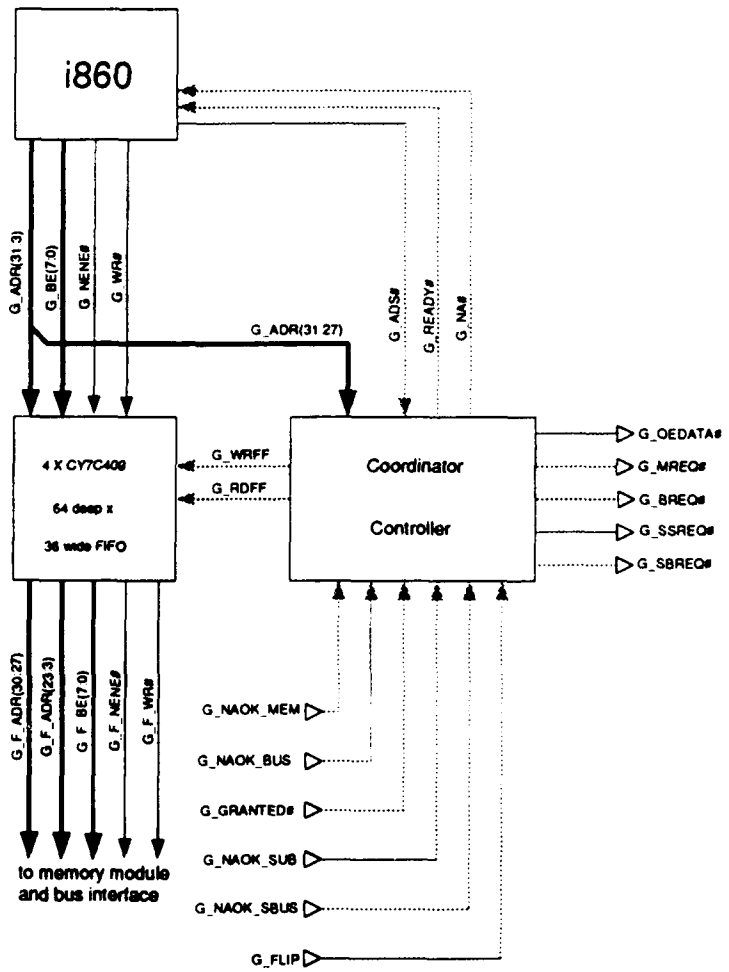


Figure 3. Master coordinator block diagram

The coordinator controller of Figure 3 is comprised of 3 state machines, each of which will now be described in turn.

The FIFO state machine (Figure 4) is used to keep track of how many outstanding cycles are stored in the address FIFOs. The machine also implements a 3-deep 2-bit wide synchronous FIFO which is used to store the cycle type of the corresponding address in the address FIFO chips. Although the address FIFO chips themselves could have been used for this purpose, they were found to be too slow to allow the fastest timings to be used.

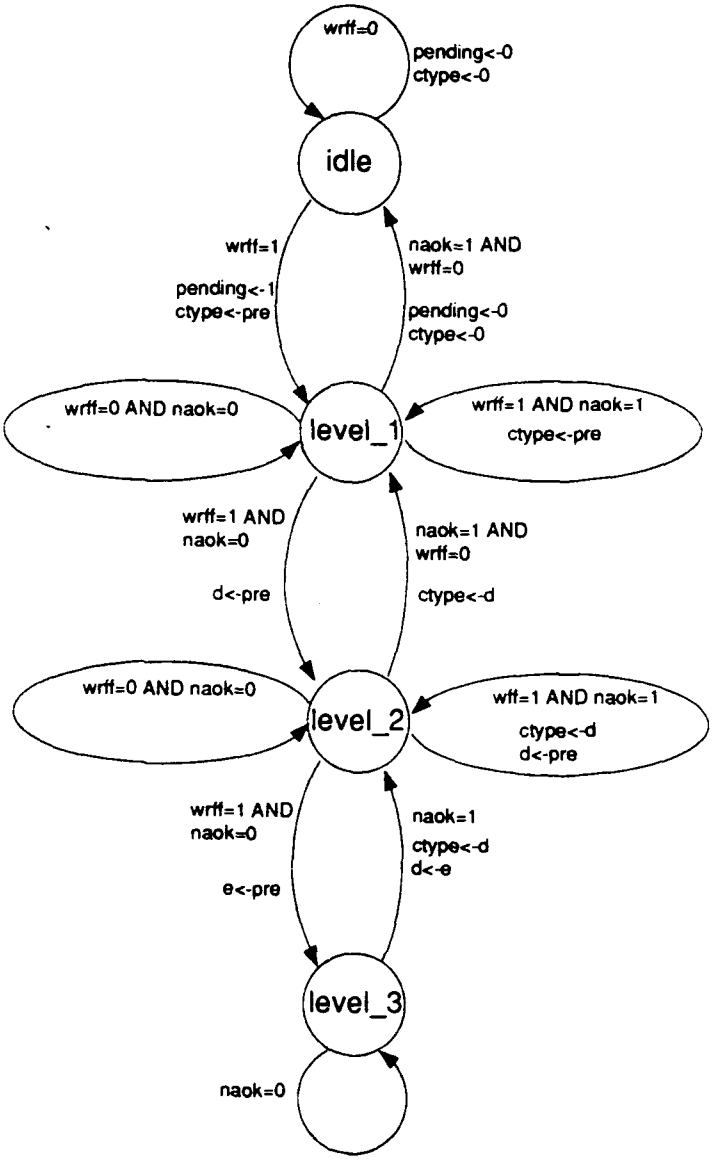


Figure 4. Cycle type FIFO state machine

Figure 5 shows the READY generator state machine. This machine listens for a NAOK from one of the modules and then produces a READY# to the i860 after an appropriate delay. For reads across the bus the delay is 3 cycles, for hotline reads it is 2 cycles and for all other accesses READY# comes on the cycle after NAOK. The extra delays are required for the reads to allow the data to propagate through the sets of transceivers involved.

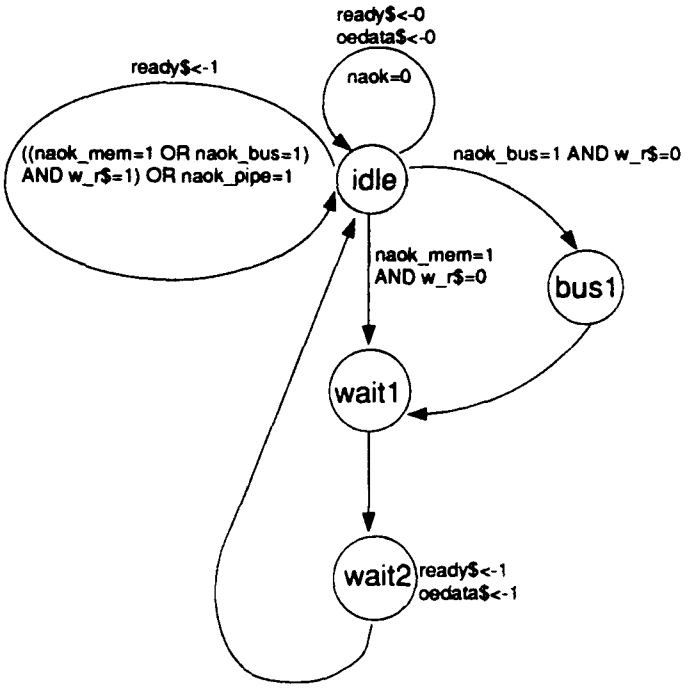


Figure 5. READY generator state machine

The main state machine in the coordinator is the request control state machine (Figure 6). This machine is responsible for determining what type of external cycle is required and generating the signal that activate the required module. For example, if the address generated by the i860 indicated a hotline access, the coordinator would set MREQ# active. The machine then waits for an NAOK (Next Address OK) from the accessed module and begins processing the next address (if there is one) in the address FIFOs. Note that there may be some overlap of cycle processing. For example, when processing consecutive memory reads, the NAOK from the memory module comes 3 cycles before the read data is supplied (corresponding to READY# being active). If the next memory cycle is initiated soon after NAOK_MEM, then this cycle will overlap the previous one. In this way maximum bandwidth can be obtained from the DRAM modules.

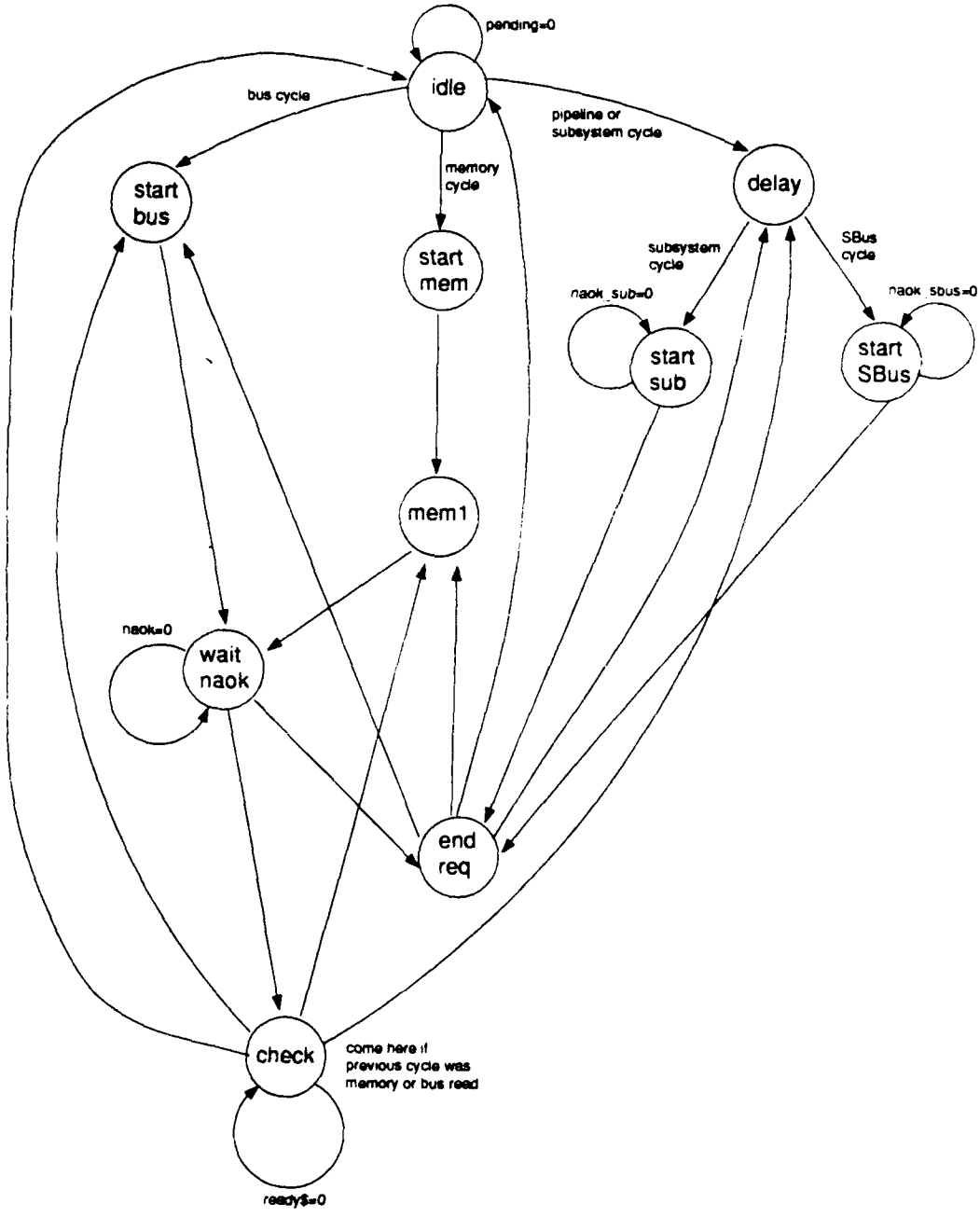


Figure 6. Request control state machine

3.1.2 Memory

The memory unit is based on two 2M × 36-bit DRAM modules, and therefore has a capacity of 16 MBytes (including checkbits). A single flow-through EDAC (Error Detection And Correction) chip implements one-bit error correction, two-bit error detection. There is one memory unit per slave and one for the master. The general organisation is depicted in Figure 7.

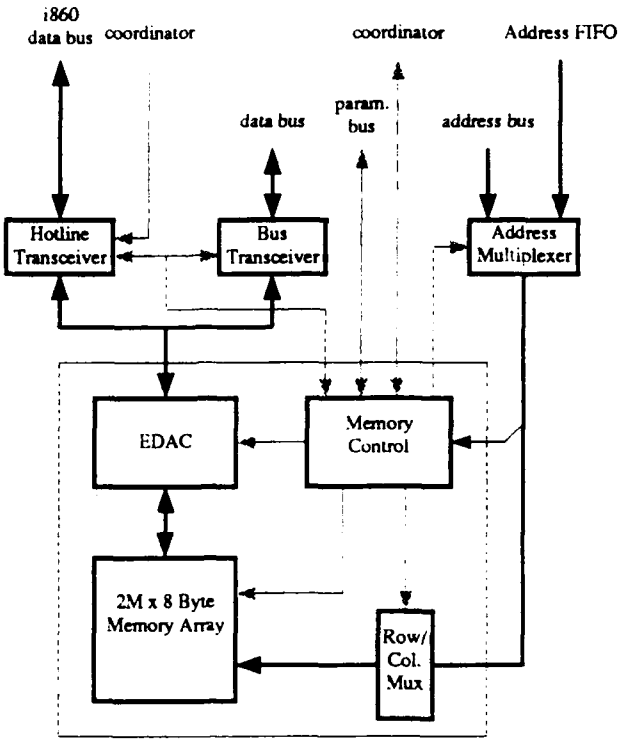


Figure 7. Memory Unit

Memory access.

The memory unit can be accessed directly by the local i860 (hotline access) or through the bus; a local priority bit, which is toggled after each full cycle, ensures that each port receives equal treatment.

A memory access is initiated by either of the request lines becoming active (H_REQ# from the local coordinator or B_REQ# from the bus arbitrator). In the case of a bus request, the local memory control decodes the higher order address bits to determine if it is being accessed; an extra cycle is needed to switch the address multiplexer by asserting AMUX which is also used to signal the arbitrator that the memory is being accessed. The operation is then carried out as follows (see also Figure 8): first, the row address strobes (RAS) are activated, then the row/column multiplexer switches to the lower 10 bits of the address (column) and the column address strobes (CAS) are activated. The relevant NAOK (bus or hotline) is asserted once the address is no longer needed.

If a new request is present at the end of the current operation and is from the same source (bus vs. hotline), is of the same nature (read vs. write), is in the same page (NENE# asserted), and no refresh is pending, then only the CAS are cycled to latch in the new (column) address. In this case, which will be referred to as "page mode", consecutive memory accesses are carried out in 4 cycles, or 100 ns[1]. Since each memory access can supply 64 bits of data, the maximum memory bandwidth is 80MB/s.

1. Some further speed-up may be achieved either by taking the EDAC off-line, which implies using a "correct-only-on-error" mode, or using parity checking instead of the EDAC. In the first case, overheads occur only when there is an error but the reliability remains the same, while in the second case any error should result in an unrecoverable interrupt.

At the end of the operation, the RAS are deasserted and a few wait cycles are needed for pre-charge before starting a new access.

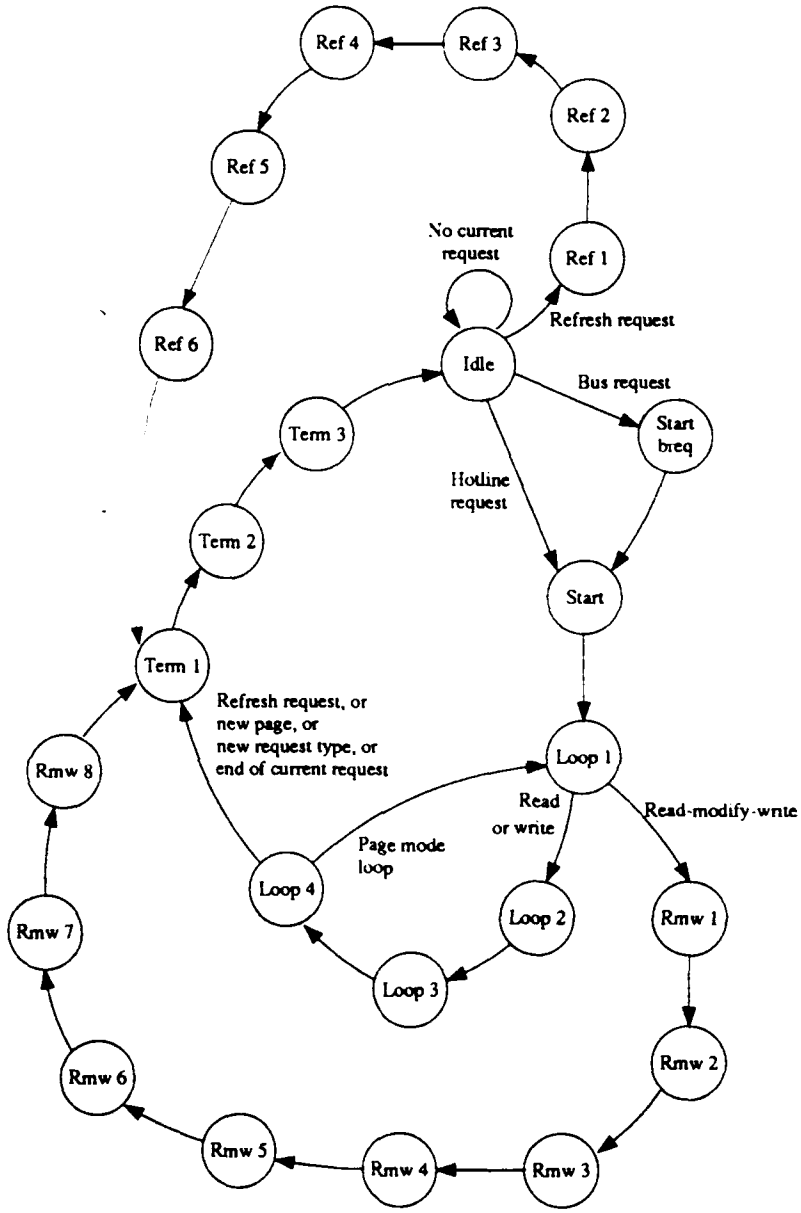


Figure 8. Memory State Diagram

Read cycle.

For simplicity, the EDAC is set to "correct-always" mode. No action is taken in case of an error, whether or not correctible, apart from lighting the corresponding LED which can only be turned off with SYSRESET. The syndrome bits cannot be read, at least in this version, as there are no diagnostics paths. The read data is available to the i860 3 cycles after NAOK (4 for bus accesses), which means that in page mode a new access is initiated by the coordinator while it is driving the data.

Write cycle.

If all byte-enable signals are active, a normal write cycle takes place. Checkbits are generated by the EDAC each time data is written to the memory. In this version there is no separate path for the checkbits. Since the memory modules use the same pins for data input as for output, this is termed an "early write" in the sense that the write-enable must be active before CAS so as to turn off the DRAM outputs.

Byte-write cycle.

If at least one byte-enable is inactive, a "read-modify-write" is implemented; while the write data is held in the transceivers, data is read from the memory. As the transceiver drives only those bytes to be written, the EDAC drives their complement onto its outputs (i.e., processor or bus side). Next, checkbits are generated for the new combination and the entire 8 bytes are written back to memory. No page mode is implemented in this version, again because of the PLDs' restricted capacity.

Refresh cycle.

A counter generates a refresh request to the memory control every approximately 13 µs[1]. The current operation is not interrupted, but a refresh takes precedence over page mode. A refresh cycle starts from the idle state; "CAS-before-RAS" refresh is implemented to make use of the DRAM chips' internal row address counter. There is no error scrubbing.

Bus access.

Figure 9 shows the data paths in case of a bus access. Signals OE_TO860_Di and OE_TO-BUS_Di are generated by the arbitrator.

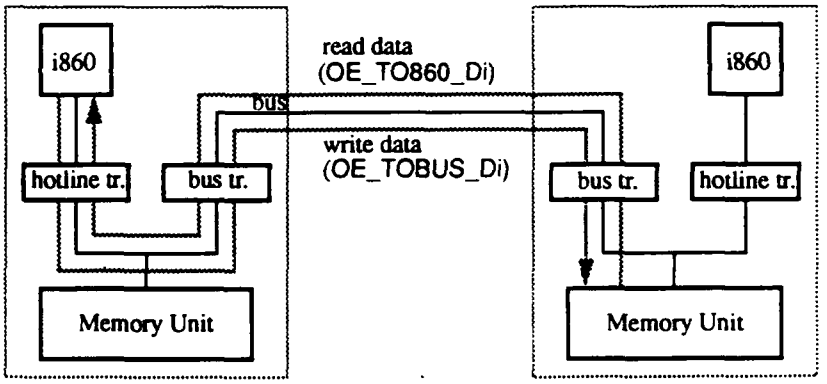


Figure 9. Bus access data paths

3.1.3 Arbitrator

In this first version, the bus arbitrator takes care of bus-memory accesses from one master unit and up to 8 slaves. This limitation is purely for design simplicity, as we would quickly run into a pin problem if we tried to implement more using our current PLDs.

While the arbitrator is located on the master board, it is functionally independent. Its purpose is to ensure fair access to the bus, at least between the slaves, by resolving contention in such a way that each processor receives equal treatment. Page mode accesses can occur but are limited to four consecutive operations, which would correspond to a block rewrite, or cache swap operation from a processor.

1. This can be changed by re-programming a PLD; depending on the manufacturer, some memory modules require less frequent refreshing.

The arbitrator handles the handshaking between coordinators and memory units, as shown in Figure 10.

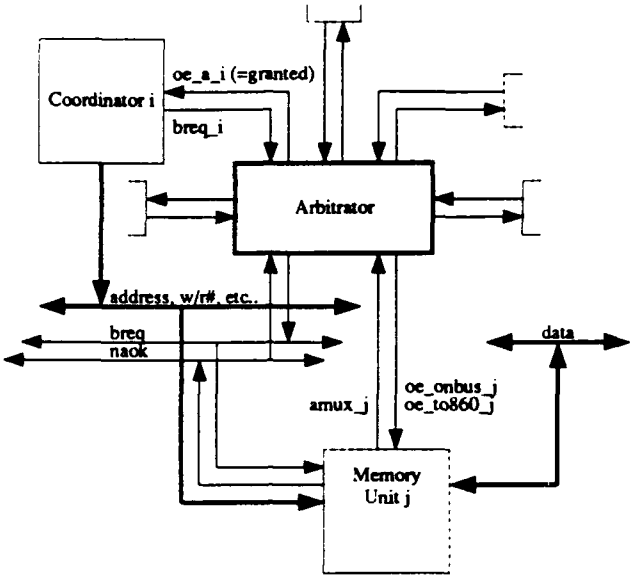


Figure 10. Arbitrator block diagram

Bus cycle

Figure 11 shows the arbitrator's state changes following a processor bus request. The arbitrator receives separate requests from each coordinator. As there can be only one bus access at a time, the arbitrator chooses the active request with the current highest priority[1] and notifies the "winner" by asserting the corresponding OE_Ai, thus driving its operation parameters onto the bus (i.e., address, byte enables, NENE# and read/write signals). As the most significant address bits represent the memory ID, the arbitrator does not need to know which memory unit is being accessed (the memory units themselves determine if the request is for them).

Next, the arbitrator drives the bus request signal (BREQ#) and waits for an acknowledge in the form of B_NAOK# being driven low. It is in fact an early acknowledge in the sense that NAOK signals the coordinator that it can shift out a new address, and start a new operation, even if one is still in progress.

In case of a write operation, the arbitrator drives the sender's write data on the bus by asserting OE_TOBUS_Di at the start of the operation, which signals the sender's local memory unit to drive data from the hotline transceiver to the bus transceiver. For a read operation, OE_TO860_Di is used upon receiving NAOK to drive the read data from the bus through to the hotline transceiver. Note that during the entire operation the local memory unit is itself inactive, only its transceivers are being used (see also Figure 9 in the section on the memory unit).

If two cycles following B_NAOK# the same coordinator requests the bus once again, it is likely that the memory is being accessed in page mode. In this case a new arbitration would take too long, as the arbitrator would have to stop driving the current parameters before driving the new ones. Instead, the arbitrator loops back to the "wait_for_NAOK" state. A two-bit counter is incremented at each loop (LEAVE) in order to force a new arbitration after four consecutive accesses from the same processor by leaving the page mode loop. This is to avoid

1. If there is no contention, the requester gets the bus regardless of its priority status.

livelock problems such as those brought about by programming errors whereby an infinite loop causes the same page to be accessed.

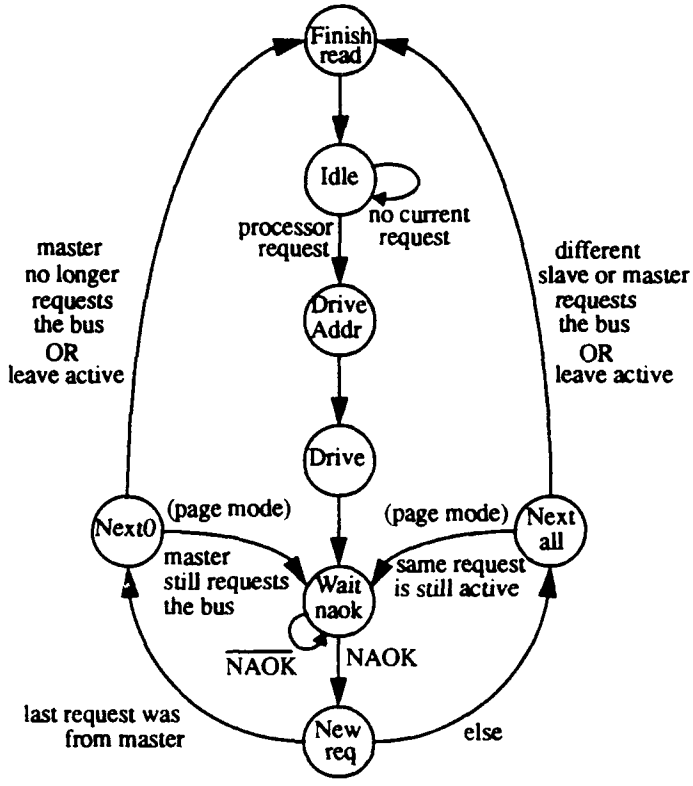


Figure 11. Arbitrator state diagram

Arbitration paradigm

Three state bits are used to determine absolute priority orderings between slave units. A fourth priority bit arbitrates between the slaves and the master, giving the latter highest priority every second access. This should not have much impact on bus contention since the master is a heavy user of the bus only at the beginning and the end of program execution. The priority bits are generated by an in-built counter which is toggled after a processor has been chosen to access the bus, therefore upon leaving the "idle" state, but not in the page loop.

In Table 1, priority order [P_i P_j P_k] means that Processor 'i' gets the bus if required, else P_j if required, else P_k if required. P-code represents the priority bit encoding for each state. The state numbers show the sequence in which the P-codes are generated. The order is arbitrary, and has been chosen as much as possible so that a processor which has had a high priority is given a low priority in the next state.

STATE	P-CODE	PRIORITY ORDER
0	000	P1 P2 P3 P4 P5 P6 P7 P8
2	001	P2 P1 P4 P3 P6 P5 P8 P7
4	010	P3 P4 P1 P2 P7 P8 P5 P6
6	011	P4 P3 P2 P1 P8 P7 P6 P5
7	100	P5 P6 P7 P8 P1 P2 P3 P4
5	101	P6 P5 P8 P7 P2 P1 P4 P3
3	110	P7 P8 P5 P6 P3 P4 P1 P2
1	111	P8 P7 P6 P5 P4 P3 P2 P1

Table 1: Arbitrator priority order

This is implemented by realising the following set of Boolean functions:

For all values of i : $granted_i = req_i \wedge \neg\{req_{j,j\neq i} \wedge Pcodes_{P_i < P_j}\}$

The disabling term in curly brackets represents the set of AND products of requests by the priority states for which their priority is greater than the enabling request's.

In Figure12, the term $\{Amux_i\}$ represents the set of signals generated by the memory units which switch their address multiplexers to the bus, thereby signifying acceptance of a bus request.

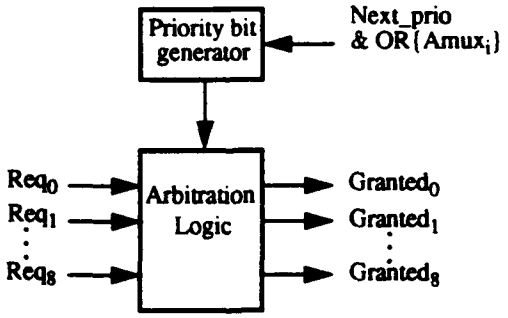


Figure 12. Arbitration logic

3.1.4 Subsystem

The subsystem consists of a number of miscellaneous functional units which are controlled from a common state machine. These units are a real time clock (useful for benchmarking and performance measuring), bootstrap ROM, RS232 (serial) interface and a number of single bit input and output registers.

Real Time Clock

The clock is based on the National Semiconductor DP8570A Timer Clock Peripheral chip. It features 12/24 hour mode timekeeping, 100th second timer resolution and a battery backup to maintain the correct time when the Shiva is powered down. The main use of the timer chip will be to generate a real time reference and to provide periodic interrupt signals to the i860.

Bootstrap ROM

The ROM used is a 256K × 8 bit Intel 27C020. It is used to store the reset bootstrap program. When accessing the ROM the i860 operates in CS8 mode which means that a usual 64-bit code fetch is broken down into consecutive 8-bit fetches from the ROM and the byte enables BE2-BE0 act as the least significant address bits A2-A0.

The current bootstrap program loads a monitor program into the system DRAM and then jumps to the start of this program.

RS232 Interface

The serial port on the master is based on the Intel M82510 Asynchronous Serial Controller. It is used to provide a console port to the Shiva via which the operator can control and monitor system operation. It is expected however, that transfers of program binaries and large data blocks will be through the much higher bandwidth SBus interface.

Registers

The subsystem provides up 24 single bit output and 8 single bit input registers. The functions of these registers are summarised in the following tables. Note that the S_INT_*n* signals are interrupts for each of the slave i860s and the G_S*n*_EAR are signals sent to the master by each slave which can be used for processor synchronisation (these signals are not subject to bus arbitration).

The LEDs can be used for general status monitoring. The functions of the remaining outputs are as follows: G_KEN# controls the cache enable on the i860, G_CS8_MAP# controls the swapping of the subsystem and the hotline image in the high end of the address map, G_ENERR controls error correction by the EDAC and SPEC_RESET# is the reset signal that controls all of the slave boards.

Physical Address	Function
0xC0180000	S_INT_1
0xC0180008	S_INT_2
0xC0180010	S_INT_3
0xC0180018	S_INT_4
0xC0180020	S_INT_5
0xC0180028	S_INT_6
0xC0180030	S_INT_7
0xC0180038	S_INT_8

Table 2: Output Register 0

Physical Address	Function
0xC0200000	G_KEN#
0xC0200008	G_CS8_MAP#
0xC0200010	G_ENERR
0xC0200018	SPEC_RESET#
0xC0200020	reserved
0xC0200028	reserved
0xC0200030	reserved
0xC0200038	reserved

Table 3: Output Register 1

Physical Address	Function
0xC0300000	LED0
0xC0300008	LED1
0xC0300010	LED2
0xC0300018	LED3
0xC0300020	LED4
0xC0300028	LED5
0xC0300030	LED6
0xC0300038	LED7

Table 4: Output Register 2

Physical Address	Function
0xC0100000	G_S1_EAR
0xC0100008	G_S2_EAR
0xC0100010	G_S3_EAR
0xC0100018	G_S4_EAR
0xC0100020	G_S5_EAR
0xC0100028	G_S6_EAR
0xC0100030	G_S7_EAR
0xC0100038	G_S8_EAR

Table 5: Input Multiplexer

3.1.5 SBus Interface

The SBus interface is split between the Master Unit and a small card designed as per specification for use in a SPARCstation [8]. Figure 13 shows that control of SBus accesses is divided into separate state machines with an asynchronous handshaking protocol between them, as the SBus card is clocked by the SPARCstation and hence is not synchronised with the Shiva.

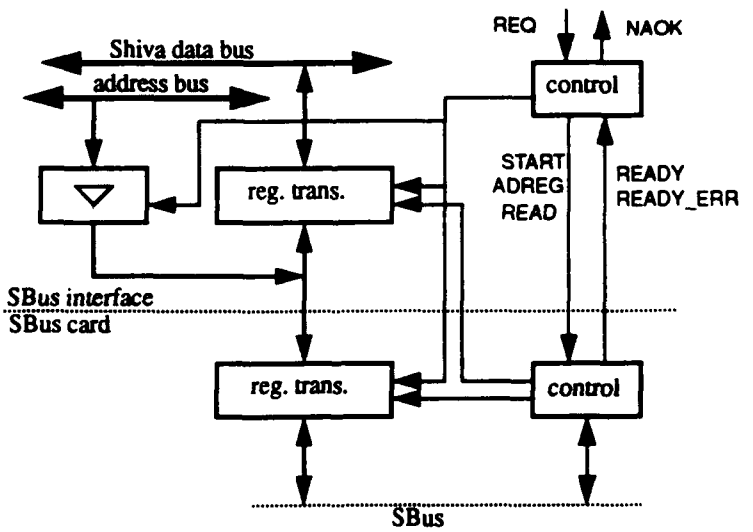


Figure 13. SBus interface

SBus protocol.

The SBus is fully synchronous and is operated by 3 types of devices: a controller, master(s) and slave(s). The data path is 32 bits wide. Several types of transfers can be carried out to or

from the SBus, from single-byte to 64-byte block transfers. As the latter would require some multiplexing and a direct memory access path to achieve maximum throughput, it was decided to implement only 4-byte transfers for simplicity reasons.

A "master" accesses the SBus by asserting a request signal, waits for it to be granted by the SBus controller, and then places the virtual address on the bus for exactly one cycle followed by data in case of a write operation. The address bus is used by the controller for the physical address. The master then waits for an acknowledgement which signals that the data, in case of a read operation, will be on the bus during the next cycle. The operation parameters (Read/ Write etc...) are to be driven by the master at the same time as the virtual address until one cycle after the acknowledge has been received.

Access from the Shiva.

Four-byte read and write operations can be effected from the Shiva. However, since the SBus data bus is also used for the virtual address, initial loading on the SBus card of the virtual address, or part of it, is necessary. From the Shiva side, however, this is considered as a write operation when address bit A29 is set, which causes ADREG to be asserted. The devices used on both the Shiva and the SBus card are registered bi-directional transceivers which allow data to be transmitted either directly from one port to the other or through a register. This feature is used on the SBus card where the virtual address is held in the register while the write data is sent over the direct path.

The handshaking protocol as seen from Shiva is as follows (see also Figure14): upon receiving a request from the coordinator, the interface drives the lower 12 address bits down to the SBus card, loads them into the corresponding section of the card's register, asserts READ or ADREG if required, sets START and waits for READY from the card. In case of a write operation, the interface also loads the lower 4 bytes from the i860 data bus into its registers. NAOK is asserted when READY is detected, START is de-asserted, and for a read operation the interface drives the data up from the card, through the direct path of its transceivers onto the i860 data bus.

Upon receiving START, the SBus card reacts as follows: if ADREG is asserted (i.e., base address operation), the data is driven down to the card, the higher 20 bits are loaded into the register and READY is asserted for one cycle. Therefore the base address operation does not involve the SBus. For normal operations, the SBus protocol previously described is followed, using the base address and the lower 12 address bits as virtual address. The purpose of this scheme is to avoid having to carry out a base address operation each time the SBus is accessed. Instead, it is required only if the access is outside a 4 KByte page boundary. For a write operation, the data is driven down to the card and onto the SBus immediately following the virtual address, after the bus has been granted. READY is asserted following reception of an acknowledge signal (or READY_ERR in case of an error acknowledge). Lastly, for a read operation the data is loaded into the card's register following reception of the acknowledge.

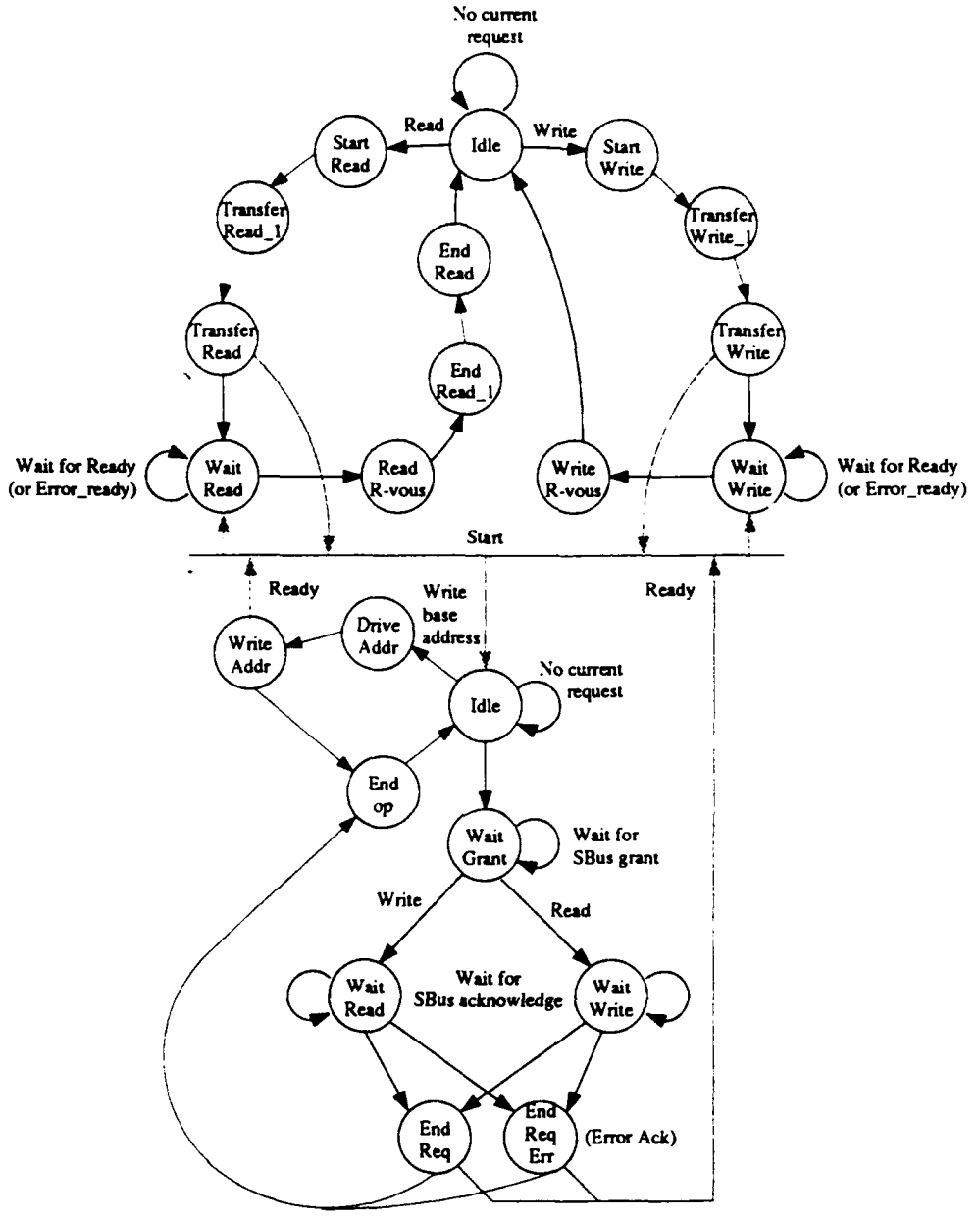


Figure 14. SBus state diagram

3.2 Slave Unit

As mentioned before, the only type of slave unit to be described in this document is one based on the Intel i860 microprocessor. This type of slave consists of the following functional units: coordinator, data pipeline, memory and bus interface.

3.2.1 Coordinator

The coordinator used in the slave is very similar to that used in the master. The only differences are that the slave coordinator does not have an SBus or subsystem interface but does

have an interface to the data pipeline. A block diagram of the slave coordinator is shown in Figure 15.

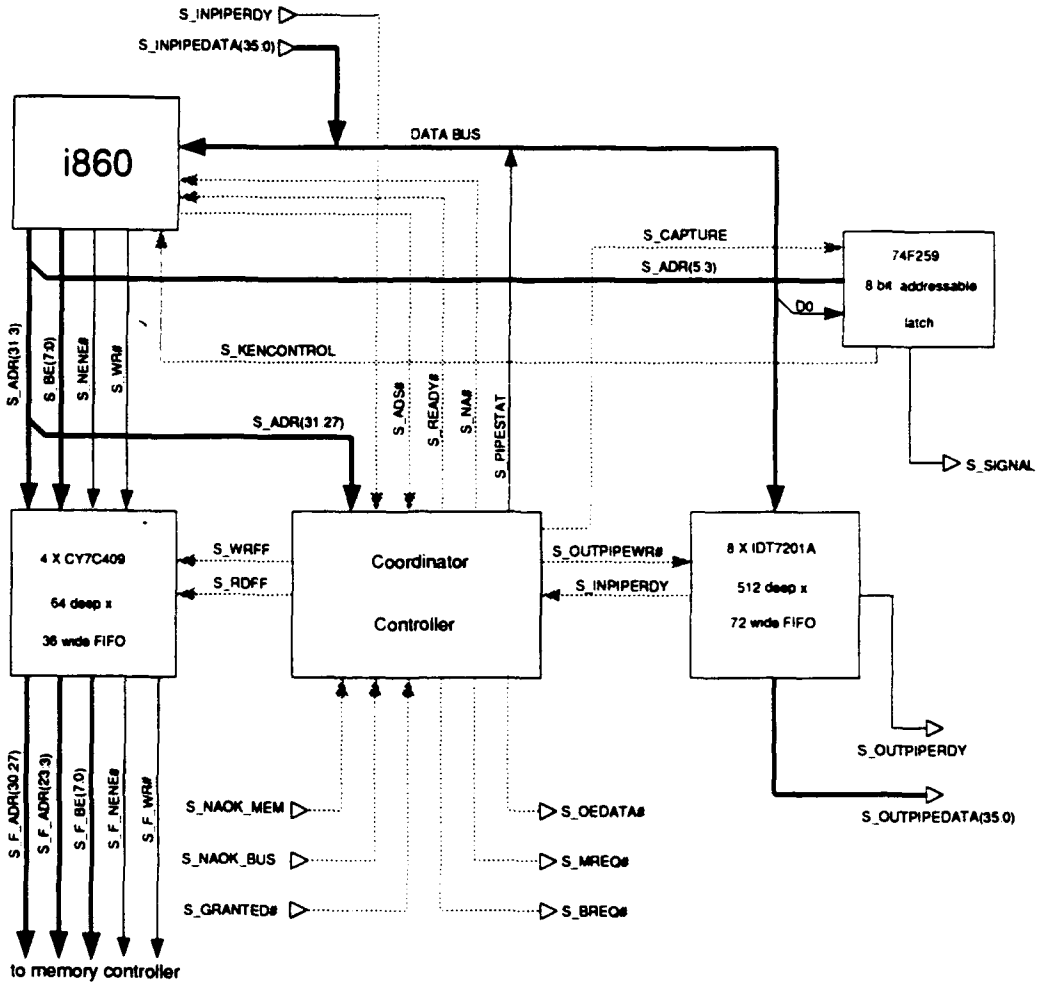


Figure 15. Slave coordinator block diagram

The coordinator controller consists of 3 state machines. The FIFO and READY generator state machines are the same as those shown in Figures 4 and 5 respectively. The request control state machine is slightly different and is shown in Figure 16. This state machine controls the pipeline and also a built in subsystem which consists of 8 individually addressable single bit registers.

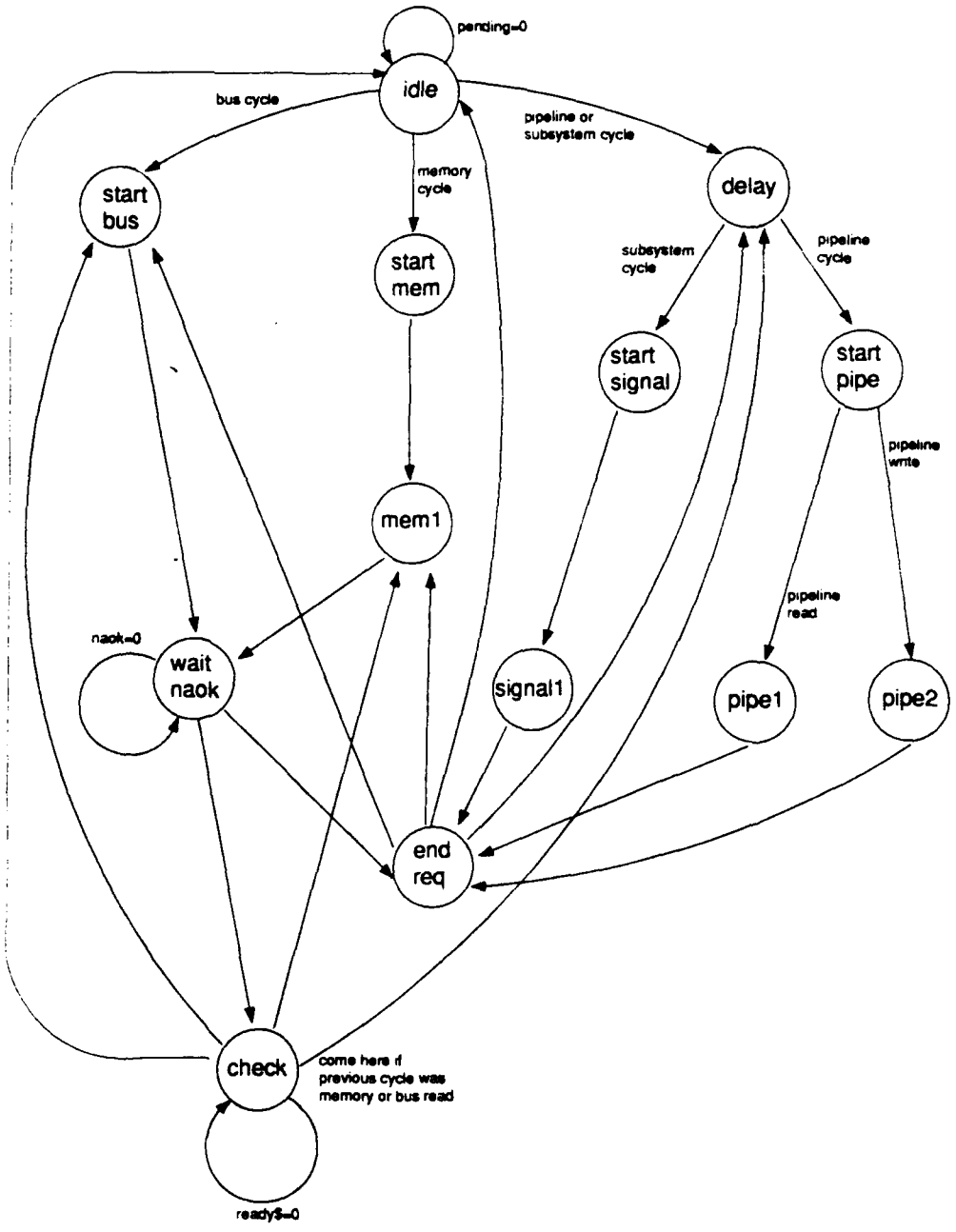


Figure 16. Slave request control state machine

The memory map for a slave is as shown in Figure 17. It should be noted that the physical address of any particular memory location is the same for each slave and the master PE, and so a process does not need to know which PE it is executing on in order to access any particular item of data in any memory unit.

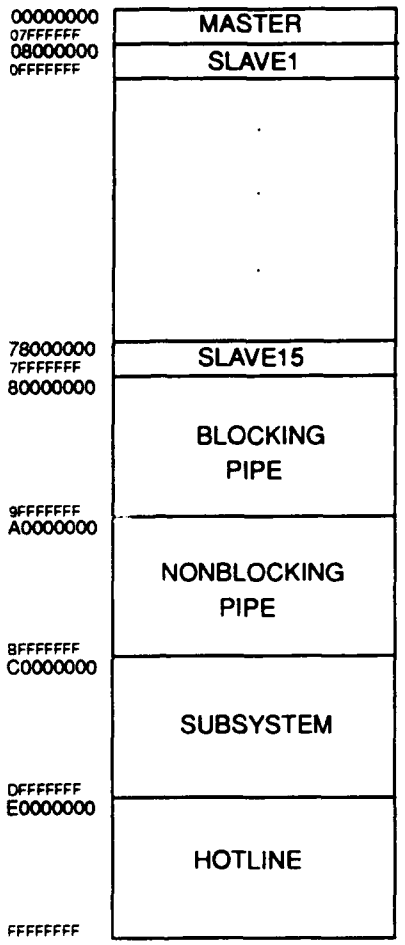


Figure 17. Slave Unit memory map

3.2.2 Data Pipeline

One of the more novel aspects of the Shiva architecture is the inter-slave data pipeline. This, together with the shared bus, provide two mechanisms for interprocessor communication. Unlike the bus, the data pipeline is contention free, i.e., all of the slave units can be writing to their data pipeline simultaneously. The advantages of such a communication mechanism are discussed in [3].

The pipeline is 64 bits wide and can support a write (and a read) every 4 clock cycles. This implies a peak bandwidth of 80MB/s which is the same as the peak memory and bus bandwidth. There are two modes for accessing the pipe: blocking and non-blocking. With a blocking access the requesting processor will be suspended if it attempts to read from an empty FIFO buffer or write to a full FIFO buffer (the buffer is 512 words deep). A non-blocking access will not suspend on a read from an empty buffer or a write from a full buffer. An attempt to write to a full buffer will result in the write data being lost and an attempt to read from an empty buffer will result in undefined data being returned. It is up the controlling software to determine when it is appropriate to perform non-blocking pipeline operations.

REFERENCES

[1] Watson C.R. & Cavaiuolo M., *A Neural Accelerator*, Proceedings, 9th Australian Microelectronics Conference, Adelaide, 1990.

[2] Anderson M., Yesberg J., Yakovleff A., Krnak D., Drewer P., Watson C., Cavaiuolo M., , *A Heterogeneous Parallel Accelerator for Image Analysis and Radar Signal Processing*, Proceedings, 25th Hawaii International Conference on System Sciences, 1992.

[3] Anderson M. & Drewer P., *Design Overview of the Shiva*, Proceedings, IEEE Region X Conference, Hong-Kong, 1990.

[4] Yakovleff A., Yesberg J., Krnak D., Anderson M., Drewer P., *An expandable supercomputer architecture with dynamic reconfigurability properties*, Proceedings, 4th Australian Supercomputer Conference, Gold Coast, 1991.

[5] Krnak D., Yakovleff A., Yesberg J., Wanless B., Anderson M., Drewer P., *Shiva Mark I - Detailed Hardware Design*, ITD Divisional Paper, ITD-91-13, 1992.

[6] Neal Margulis, *i860 Microprocessor Architecture*, Osborne, 1990.

[7] *i860 Hardware Reference Manual*, Intel, 1990.

[8] *SBus Specification A.2*, Sun Microsystems, 1990.

Appendix A· TIMING DIAGRAMS

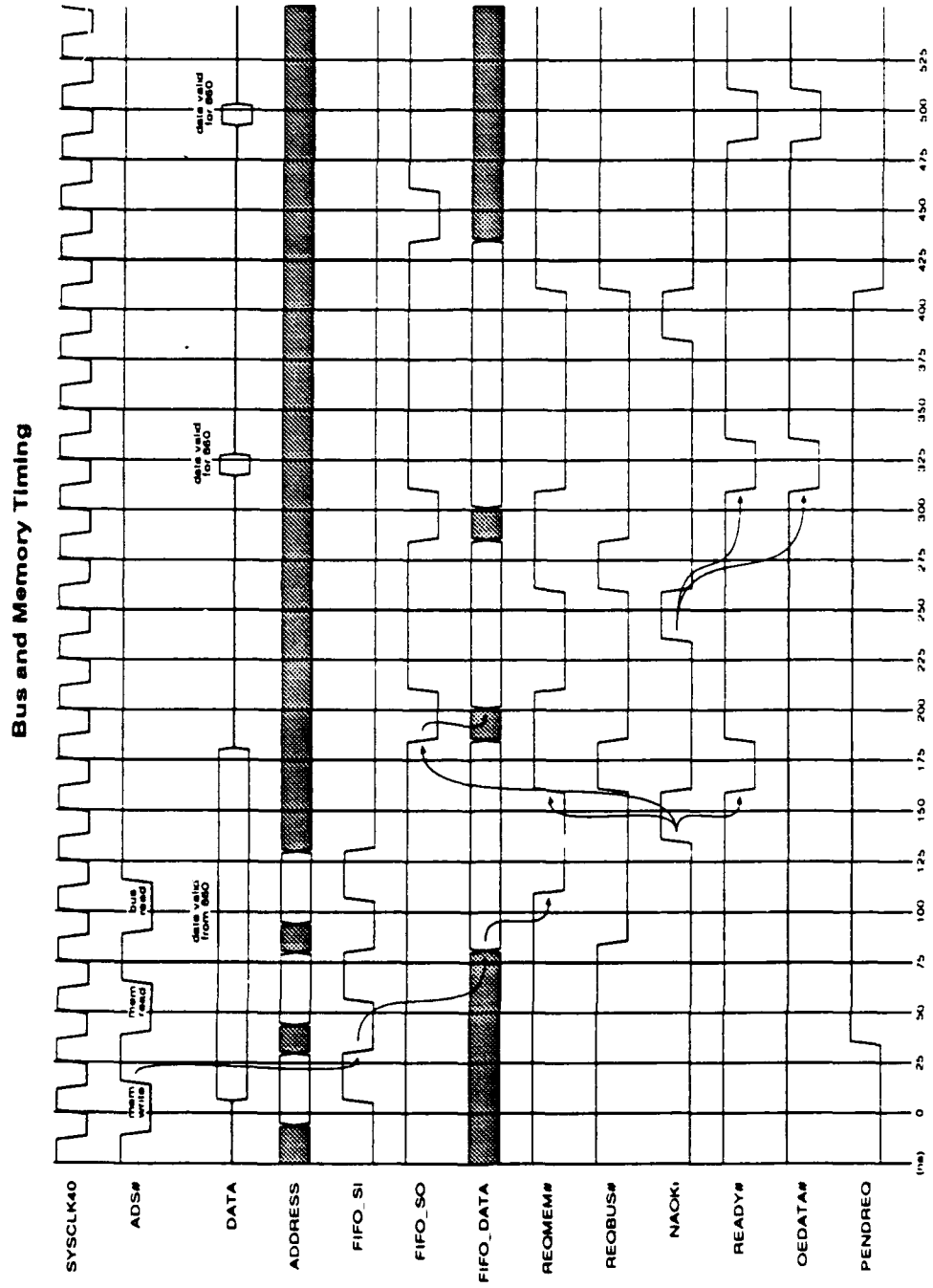
This appendix presents all the timing diagrams for the Shiva Mark II. Further details of the i860 signals and timing can be found in [7].

Glossary of Signal Names

SYSCLK40	system 40 MHz clock which goes to all synchronous components
ADS#	active low when the i860 initiates a new bus cycle (and address and byte enables are valid)
DATA	indicates when data from the i860 is valid (in the case of a write) or when the data supplied to the i860 must be valid (in the case of a read)
ADDRESS	represents when the address (including BE7-0, NENE and WR) from the i860 is valid
FIFO_SI	shift in signal to the address FIFOs
FIFO_SO	Shift out signal to the address FIFOs
FIFO_DATA	represents when the address (including BE7-0, NENE and WR) from the address FIFOs is valid
REQMEM#	active low signal indicating that the current request is for the hotline memory
REQBUS#	active low signal indicating that the current request is for the bus
NAOKi	represents the logical OR of all of the Next Address OK (NAOK) signals
READY#	active low signal that indicates to the i860 that the current cycle has completed (and that data is valid in the case of a read)
OEDATA#	active low signal controlling transceivers which drive data towards the i860
PENDREQ	indicates whether or not any requests are waiting in the address FIFOs
RAS# & CAS#	memory row and column address strobes
RCMUX	memory address multiplexer switch (when active, the column part of the address is fed to the memory)
EDAC_DIR#	data direction in the EDAC: when active, data flows to the memory
EDAC_OE#	EDAC output enable (depends on direction)
EDAC_LE	latch enable in internal EDAC register
REFREQ	memory refresh request generated by refresh counter
ENDREF	reset refresh counter signal

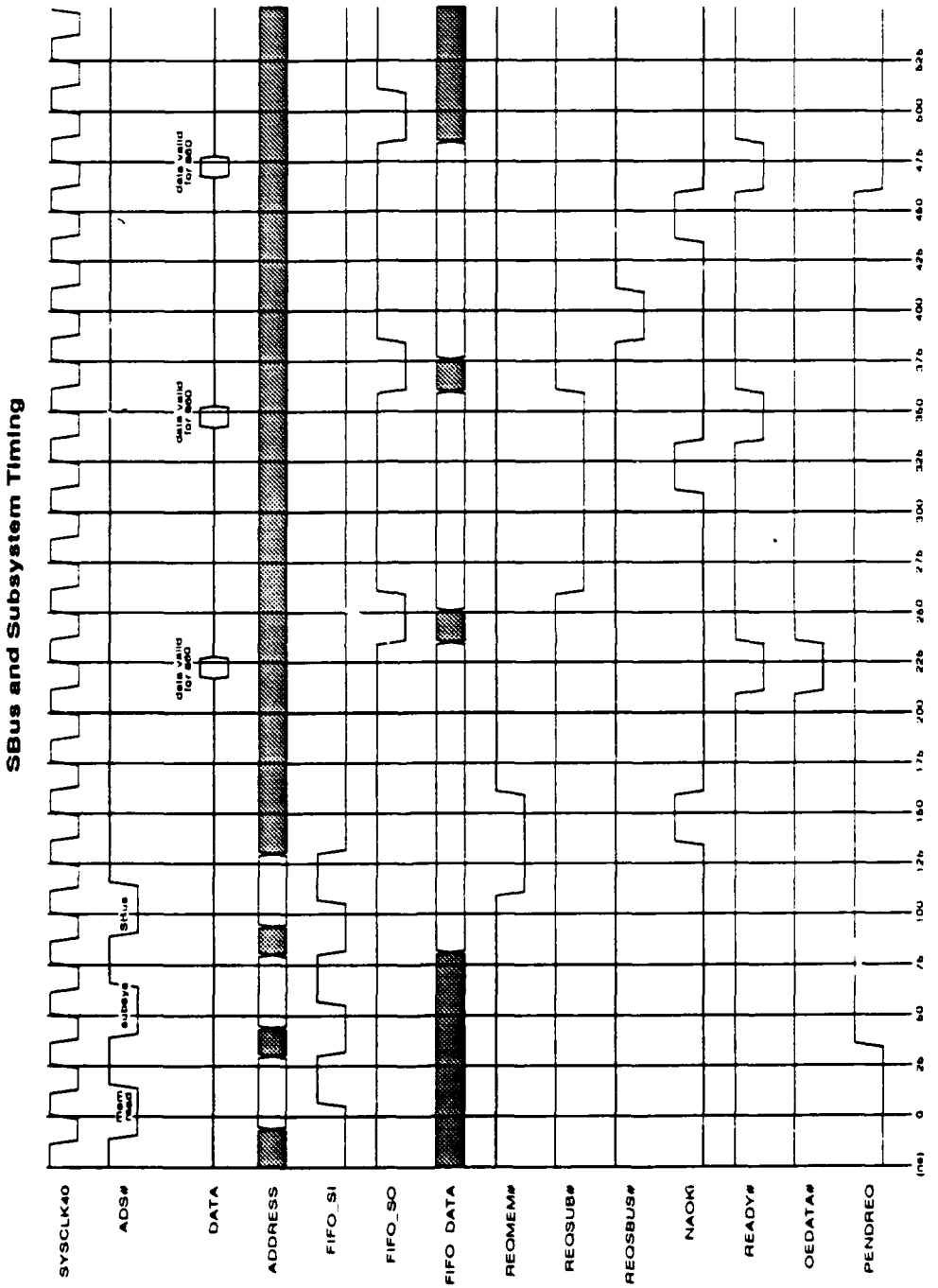
A.1 Bus and Memory Timing

The following diagram shows the timing for a hotline write (which has the same timing as a bus write), a hotline read and a bus read. Note that PENDREQ is a signal used internally by the coordinator which indicates whether or not a request is waiting in the address FIFOs.



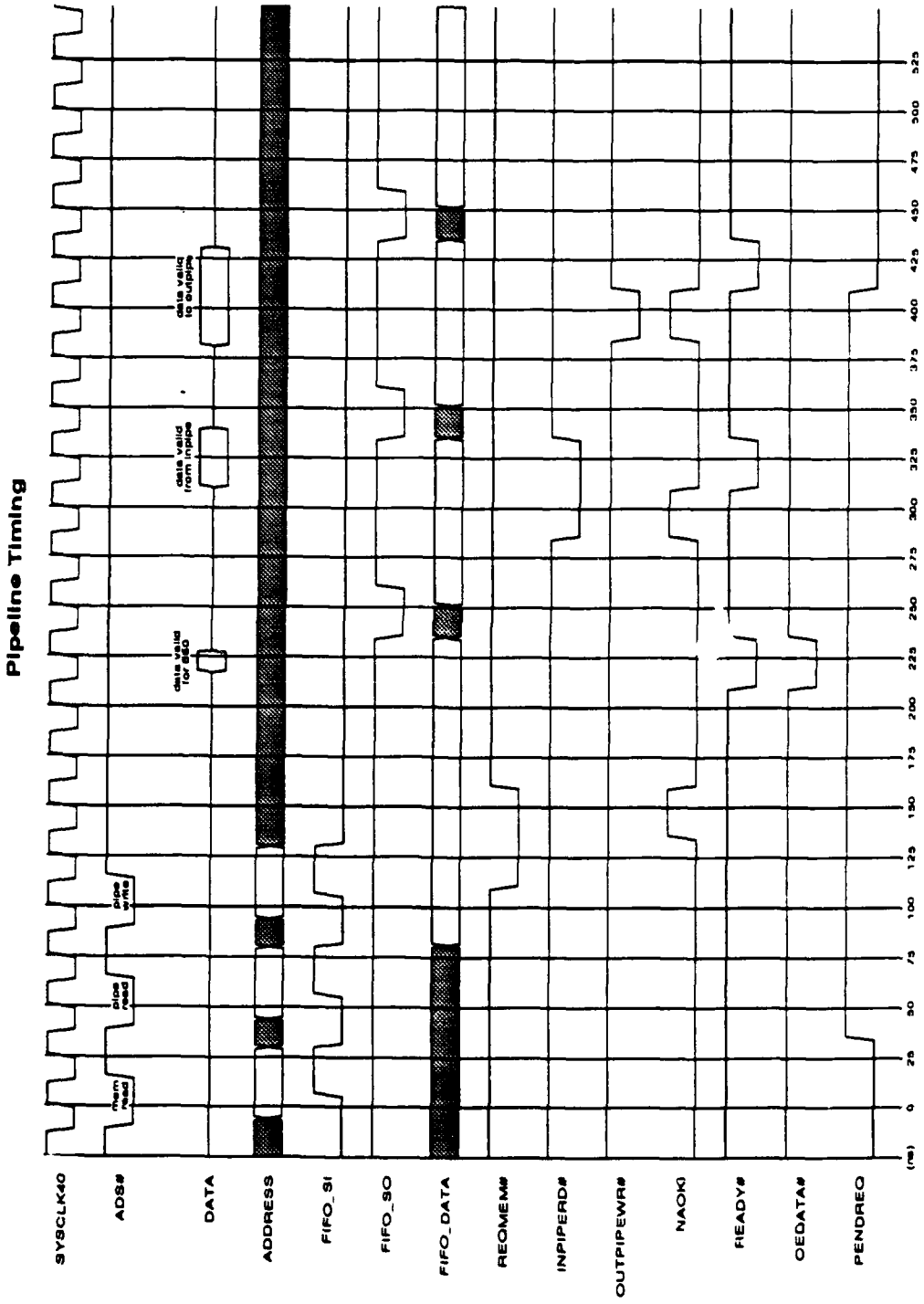
A.2 SBus and Subsystem Timing

The following diagram shows the timing for the master subsystem and SBus accesses.



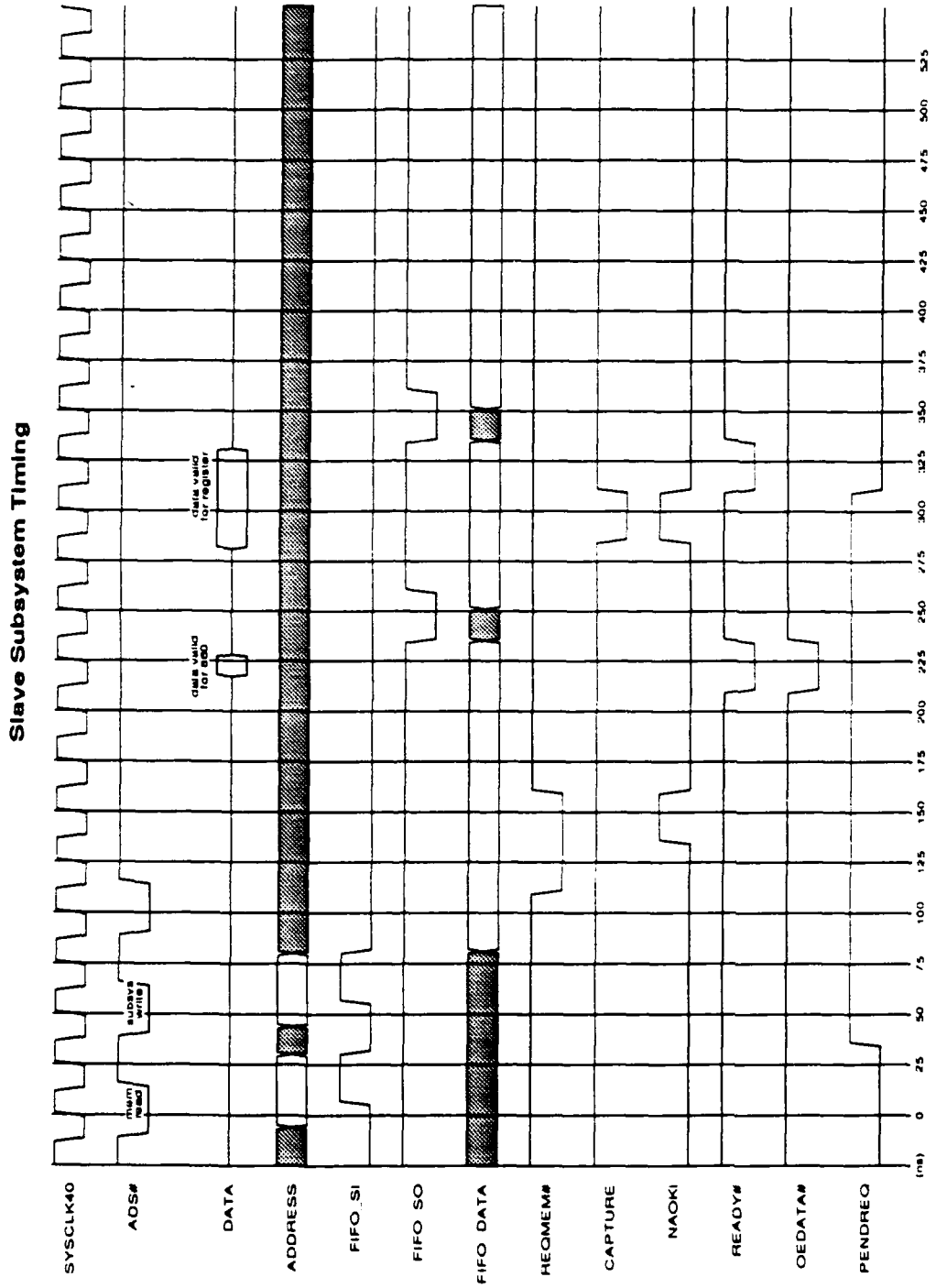
A.3 Pipeline Timing

The following diagram shows the timing for data pipeline reads and writes.



A.4 Slave Subsystem Timing

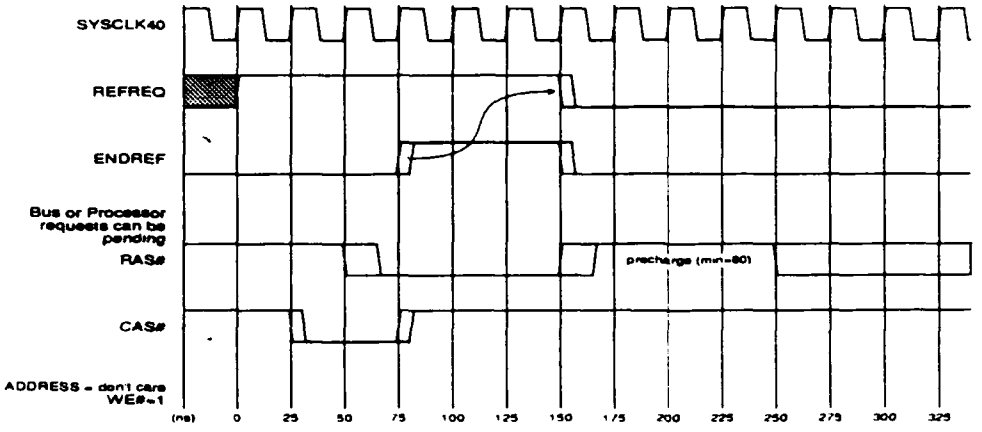
The following diagram shows the timing for the slave subsystem.



A.5 Memory Refresh Timing

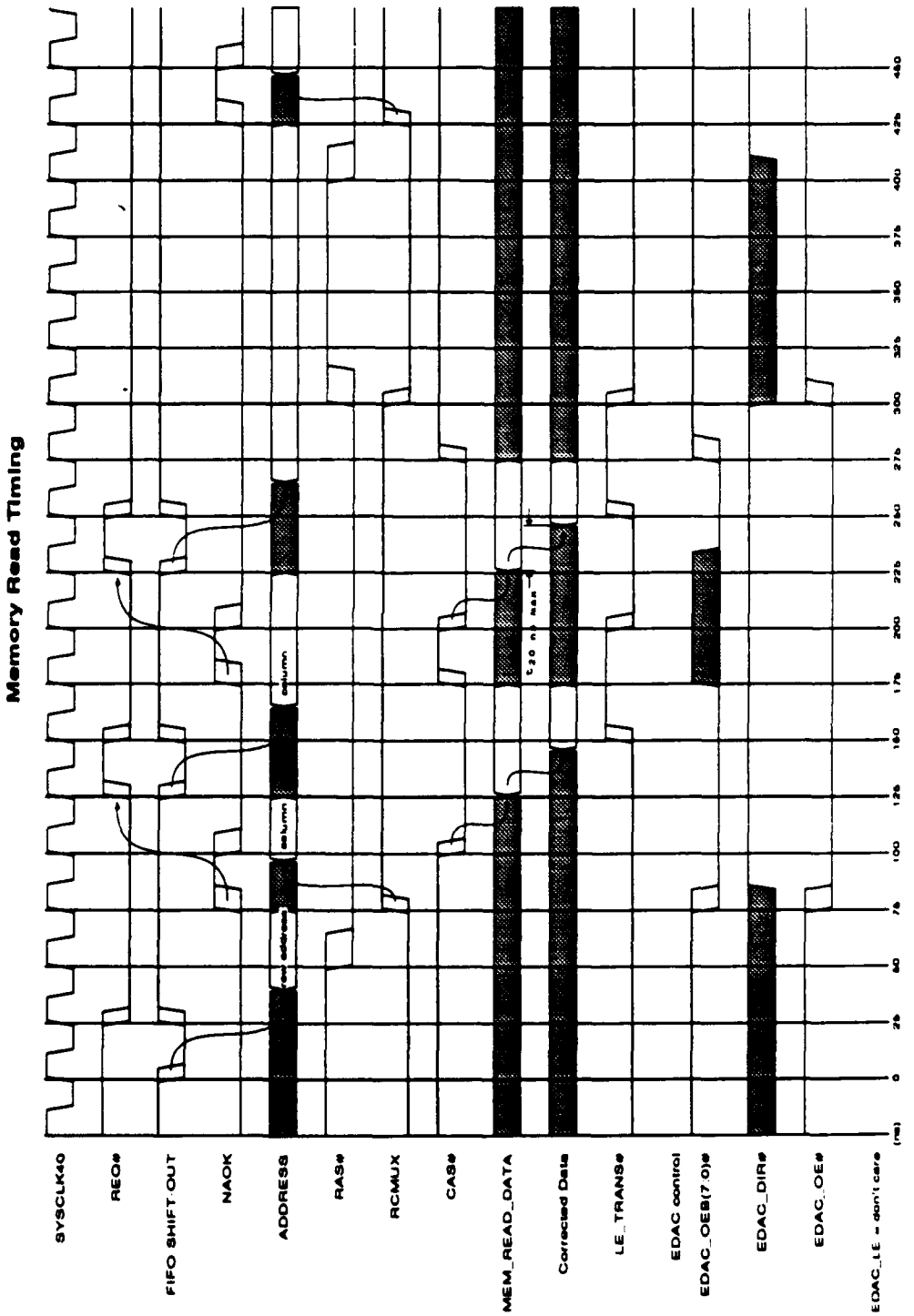
The "CAS-before-RAS" scheme makes use of the memory chips' internal refresh address counters. No NAOK is generated and a refresh cycle is known only to the memory control. When REFREQ occurs, the current operation (Read, Write or Read-modify-write) is completed irrespective of page mode, the refresh cycle is carried out and a new operation can begin.

Memory Refresh Timing (CAS-before-RAS)



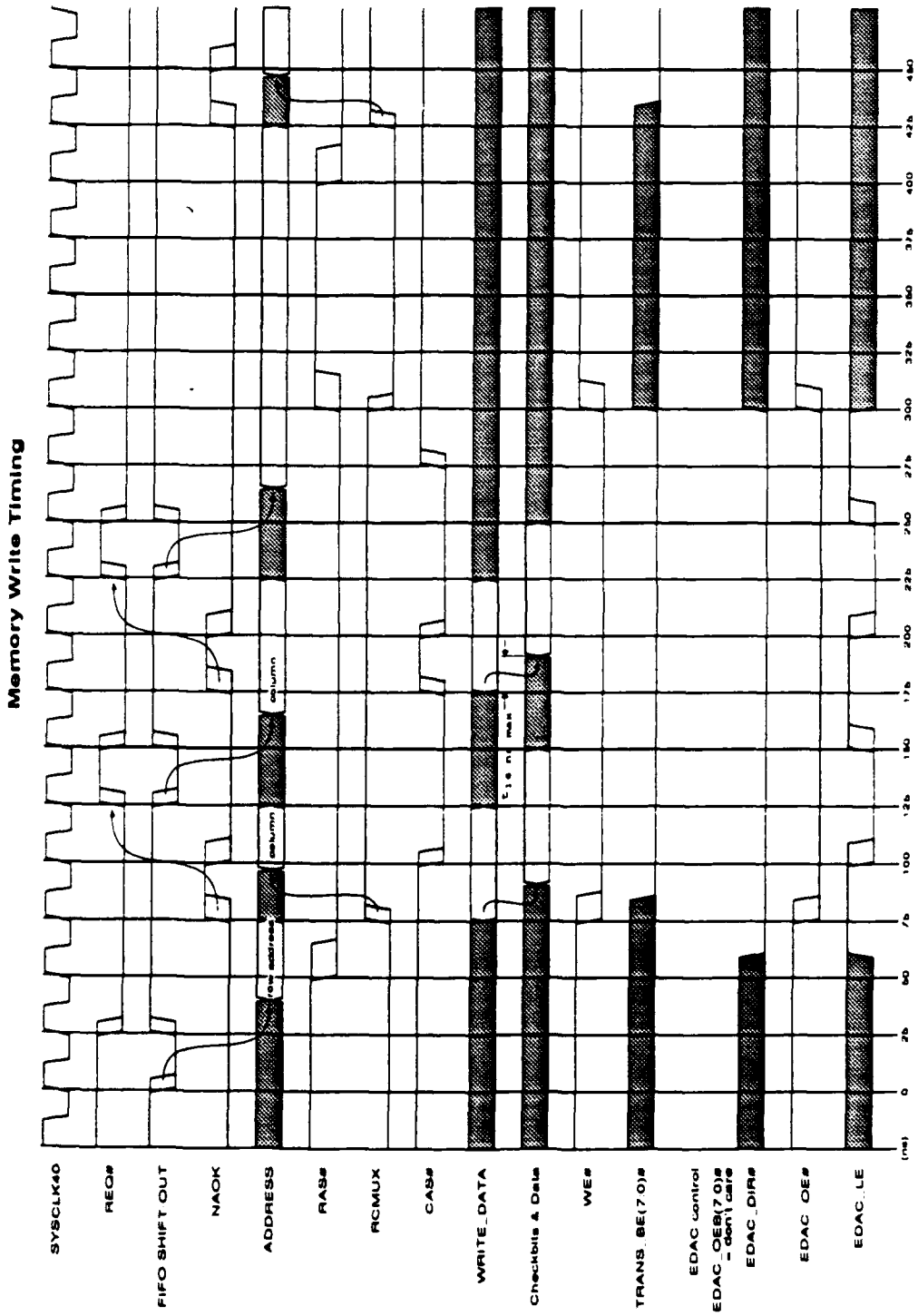
A.6 Memory Read Timing

For bus requests, one extra cycle is needed before RAS# goes down so as to switch the address.



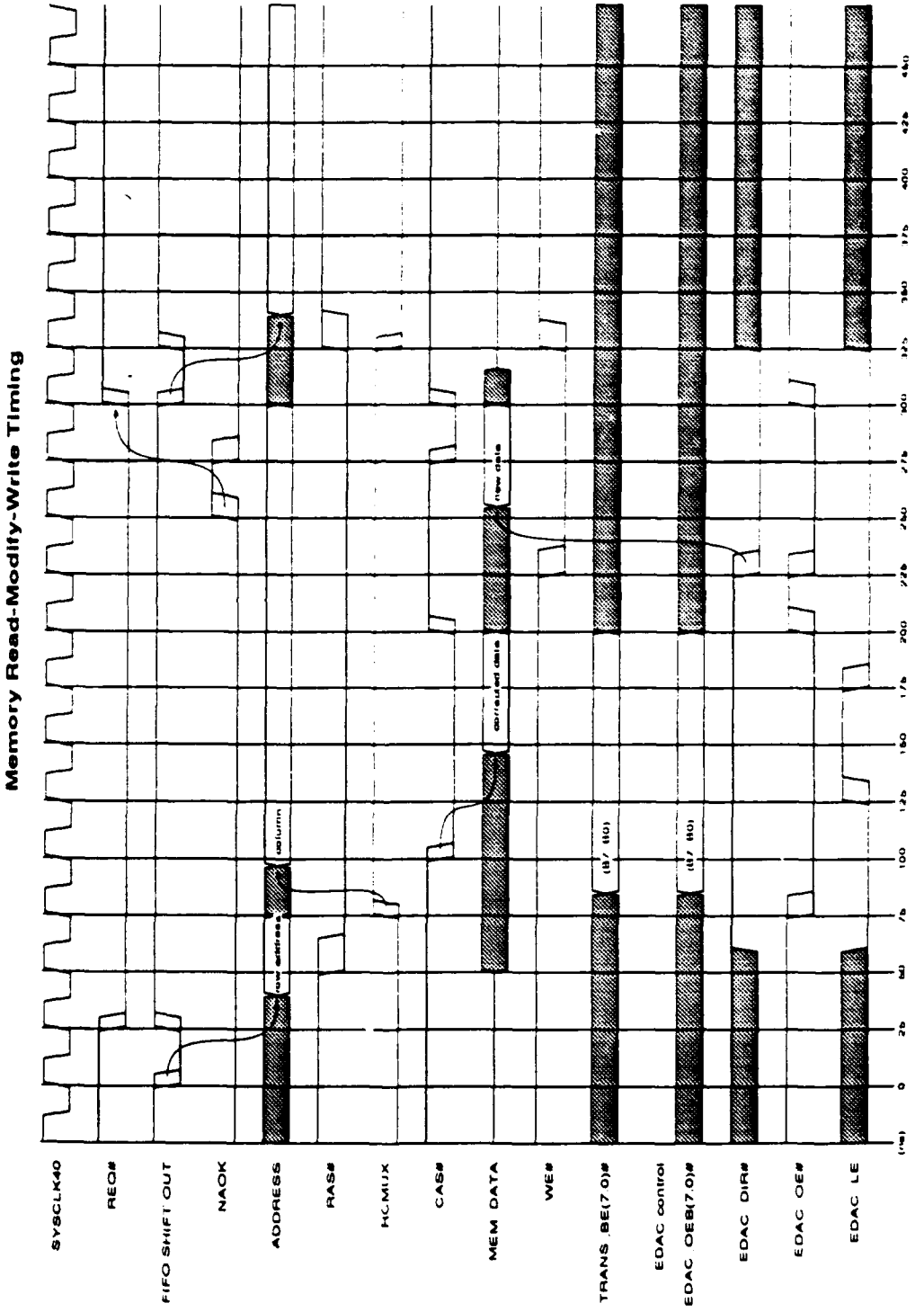
A.7 Memory Write Timing

For bus requests, one extra cycle is needed before RAS# goes down so as to switch the address.



A.8 Read-Modify-Write Timing

For bus requests, one extra cycle is needed before RAS# goes down so as to switch the address.



DISTRIBUTION

	Copy No.
Defence Science and Technology Organisation	
Chief Defence Scientist)	
Central Office Executive)	1 shared copy
Counsellor, Defence Science, London	Doc Cont Data Sht
Counsellor, Defence Science, Washington	Doc Cont Data Sht
Scientific Adviser, Defence Central	1
Scientific Adviser, Defence Intelligence Organisation	1
Navy Scientific Adviser	1
Air Force Scientific Adviser	1
Scientific Adviser, Army	1
Electronics Research Laboratory	
Director	1
Chief, Information Technology Division	1
Chief, Electronic Warfare Division	Doc Cont Data Sht
Chief, Communications Division	Doc Cont Data Sht
Research Leader Command and Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Human Computer Interaction	1
Head Computer Systems Architecture Group	1
Mr A. Yakovleff (CSA) Author	1
Mr P. Drewer (ITD) Author	1
Dr M. Anderson (TCS) Author	1
Mr J. Yesberg (TCS) Author	1
Mr P. Deer (IAP)	1
Dr M. Nelson (IAP)	1
Publications and Publicity Officer ITD	1
Media Services	1
Libraries and Information Services	
Australian Government Publishing Service	1
Defence Central Library, Technical Reports Centre	1
Manager, Document Exchange Centre, (for retention)	1
National Technical Information Service, United States	2
Defence Research Information Centre, United Kingdom	2
Director Scientific Information Services, Canada	1
Ministry of Defence, New Zealand	1
National Library of Australia	1
Defence Science and Technology Organisation Salisbury, Research Library	2
Library Defence Signals Directorate, Melbourne	1
British Library Document Supply Centre	1
Spares	
Defence Science and Technology Organisation Salisbury, Research Library	6

Department of Defence

DOCUMENT CONTROL DATA SHEET

Page Classification
UNCLASSIFIED
Privacy Marking/Caveat (of document)

1a. AR Number	1b. Establishment Number	2. Document Date	3. Task Number
AR-006-970	ERL-0631-GD	AUG 92	

4. Title

SHIVA MARK II HARDWARE ARCHITECTURE VERSION 1

5. Security Classification

Document	Title	Abstract
U	U	U

S (Secret) C (Confi) R (Rest) U (Unclass)

* For UNCLASSIFIED docs with a secondary distribution LIMITATION, use (L) in document box.

6. No. of Pages	34
7. No. of Refs.	8

8. Author(s)

D.A. Kmak, A.J.S. Yakovleff, J.D. Yesberg, M.S. Anderson and P.C. Drewer

9. Downgrading/Delimiting Instructions

Not to be downgraded without reference to the Director, Electronics Research Laboratory

10a. Corporate Author and Address

Electronics Research Laboratory
PO Box 1500
SALISBURY SA 5108

10b. Task Sponsor

DSTO

11. Officer/Position responsible for

Security.....

Downgrading.....N/A.....

Approval for Release.....DERL.....

12. Secondary Distribution of this Document

APPROVED FOR PUBLIC RELEASE

Any enquiries outside stated limitations should be referred through DSTIC, Defence Information Services, Department of Defence, Anzac Park West, Canberra, ACT 2600.

13a. Deliberate Announcement

No limitation

13b. Casual Announcement (for citation in other documents)

[√] No Limitation

[ ] Ref. by Author , Doc No. and date only.

14. DEFTEST Descriptors

Computer architecture, Multiprocessing, Multiprocessors
Computer programming, Computer systems hardware

15. DISCAT Subject Codes

1205, 1206

16. Abstract

This document describes the hardware aspects of the Shiva multiprocessor, which has a dynamically reconfigurable architecture and supports heterogeneity. The system is meant to be used as an accelerator for computationally intensive tasks, and is used in conjunction with a workstation.

Page Classification

UNCLASSIFIED

16. Abstract (CONT.)

17. Imprint

Electronics Research Laboratory
PO Box 1500
SALISBURY SA 5108

18. Document Series and Number	19. Cost Code	20. Type of Report and Period Covered
ERL-0631-GD	830004	GENERAL DOCUMENT

21. Computer Programs Used

N/A

22. Establishment File Reference(s)

N/A

23. Additional information (if required)